```
# Importing the NumPy library
import numpy as np
```

Add text cell

1. Creating Arrays

```
# Creating a 1D Array
arr1 = np.array([1, 2, 3, 4, 5])
print("1D Array:", arr1)
```

```
1D Array: [1 2 3 4 5]
```

```
# Creating a 2D Array (Matrix)
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print("2D Array:\n", arr2)
```

```
2D Array:
 [[1 2 3]
 [4 5 6]]
```

```
# Creating Arrays with Default Values
zeros = np.zeros((3, 3))  # 3x3 matrix of zeros
ones = np.ones((2, 2))  # 2x2 matrix of ones
identity = np.eye(3)  # 3x3 identity matrix
```

```


print("Zeros:\n", zeros)
print("Ones:\n", ones)
print("Identity Matrix:\n", identity)
```

```
Zeros:
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
Ones:
 [[1. 1.]
 [1. 1.]]
Identity Matrix:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```
# Creating an Array with a Range of Values
arr_range = np.arange(0, 10, 2)  # From 0 to 10 with step size 2
print("Range Array:", arr_range)
```

```
Range Array: [0 2 4 6 8]
```

```
# Creating an Array with Linearly Spaced Values
arr_linspace = np.linspace(0, 1, 5)  # 5 values between 0 and 1
print("Linspace Array:", arr_linspace)
```

```
Linspace Array: [0.   0.25 0.5  0.75 1.  ]
```

2. Array Shape and Reshaping

Add text cell

```
# Check Shape of an Array
print("Shape of arr2:", arr2.shape)
```

```
Shape of arr2: (2, 3)
```

```
# Reshape Array (Changing Dimensions)
reshaped = arr1.reshape((5, 1))
print("Reshaped Array:\n", reshaped)
```

```
Reshaped Array:
 [[1]
 [2]
 [3]
 [4]
 [5]]
```

```
# Flatten a Multi-dimensional Array to 1D
flattened = arr2.flatten()
print("Flattened Array:", flattened)
```

```
Flattened Array: [1 2 3 4 5 6]
```

3. Basic Array Operations

```
# Element-wise Operations
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print("Addition:", a + b)
print("Subtraction:", a - b)
print("Multiplication:", a * b)
print("Division:", a / b)
```

```
Addition: [5 7 9]
Subtraction: [-3 -3 -3]
Multiplication: [ 4 10 18]
Division: [0.25 0.4  0.5 ]
```

```
# Aggregate Functions
print("Sum:", np.sum(a))
print("Mean:", np.mean(a))
print("Standard Deviation:", np.std(a))
print("Max:", np.max(a))
print("Min:", np.min(a))
```

```
Sum: 6
Mean: 2.0
```

```
Standard Deviation: 0.816496580927726
Max: 3
Min: 1
```

4. Indexing and Slicing

Add text cell

```
# Accessing Elements
print("First Element of arr1:", arr1[0])
print("Element at (1, 2) in arr2:", arr2[1, 2])
```

```
First Element of arr1: 1
Element at (1, 2) in arr2: 6
```

```
# Slicing
print("Slice arr1[1:4]:", arr1[1:4])
```

```
Slice arr1[1:4]: [2 3 4]
```

```
# Boolean Indexing
bool_idx = arr1 > 2  # Find elements greater than 2
print("Boolean Index:", bool_idx)
print("Filtered Elements:", arr1[bool_idx])
```

```
Boolean Index: [False False  True  True  True]
Filtered Elements: [3 4 5]
```

5. Random Data Generation

```
# Generate Random Numbers
rand_arr = np.random.rand(3, 3)  # Uniform distribution between 0 and 1
print("Random Array:\n", rand_arr)
```

```
Random Array:
 [[0.11017776 0.86426099 0.68392292]
 [0.45824448 0.54852262 0.74252585]
 [0.662762   0.77387998 0.30971595]]
```

```
# Generate Random Integers
rand_int = np.random.randint(1, 10, (2, 2))  # Integers between 1 and 9
print("Random Integers:\n", rand_int)
```

```
Random Integers:
 [[5 1]
 [3 7]]
```

```
# Set Seed for Reproducibility
np.random.seed(42)
rand_seeded = np.random.rand(3)
print("Seeded Random Array:", rand_seeded)
```

```
Seeded Random Array: [0.37454012 0.95071431 0.73199394]
```

6. Handling NaN and Infinite Values

Add text cell

```
arr_with_nan = np.array([1, np.nan, 3, np.inf])
```

```
# Check for NaN and Infinite Values
print("Is NaN:", np.isnan(arr_with_nan))
print("Is Infinite:", np.isinf(arr_with_nan))
```

```
Is NaN: [False  True False False]
Is Infinite: [False False False  True]
```

```
# Replace NaN with 0
arr_cleaned = np.nan_to_num(arr_with_nan, nan=0.0, posinf=1000)
print("Cleaned Array:", arr_cleaned)
```

```
Cleaned Array: [   1.    0.    3. 1000.]
```

7. Linear Algebra Operations

```
# Matrix Multiplication
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

print("Matrix Multiplication:\n", np.dot(A, B))
```

```
Matrix Multiplication:
 [[19 22]
 [43 50]]
```

```
# Transpose of a Matrix
print("Transpose:\n", A.T)
```

```
Transpose:
 [[1 3]
 [2 4]]
```

```
# Determinant of a Matrix
print("Determinant:", np.linalg.det(A))
```

```
Determinant: -2.0000000000000004
```

```
# Inverse of a Matrix
print("Inverse:\n", np.linalg.inv(A))
```

```
Inverse:
 [[-2.   1. ]
 [ 1.5 -0.5]]
```

8. Broadcasting and Vectorization

```
# Broadcast Add text cellg a scalar to an array
arr = np.array([1, 2, 3])
print("Broadcasted Addition:", arr + 5)
```

```
Broadcasted Addition: [6 7 8]
```

```
# Vectorized Operations (Faster than loops)
arr_large = np.arange(1000000)
%timeit arr_large + 1  # Vectorized operation
```

```
777 µs ± 62.3 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
# Non-vectorized loop (for comparison)
def non_vectorized(arr):
    result = []
    for i in arr:
        result.append(i + 1)
    return np.array(result)

%timeit non_vectorized(arr_large)
```

```
237 ms ± 7.99 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

9. Saving and Loading Data

```
# Save an Array to a File
np.save('array.npy', arr1)
print("Array saved to 'array.npy'.")
```

```
Array saved to 'array.npy'.
```

```
# Load an Array from a File
loaded_arr = np.load('array.npy')
print("Loaded Array:", loaded_arr)
```

```
Loaded Array: [1 2 3 4 5]
```

```
# Save to a CSV File
np.savetxt('array.csv', arr2, delimiter=',')
print("Array saved to 'array.csv'.")
```

```
Array saved to 'array.csv'.
```

```
# Load from a CSV File
loaded_csv = np.loadtxt('array.csv', delimiter=',')
print("Loaded CSV Array:\n", loaded_csv)
```

```
Loaded CSV Array:
 [[1. 2. 3.]
 [4. 5. 6.]]
```

10. Pe ance Tips and Best Practices

Add text cell

```
# Use np.where for Conditional Logic
arr = np.array([1, 2, 3, 4, 5])
result = np.where(arr % 2 == 0, 'Even', 'Odd')
print("Conditional Logic Result:", result)
```

```
Conditional Logic Result: ['Odd' 'Even' 'Odd' 'Even' 'Odd']
```

```
# Use np.concatenate to Join Arrays
arr_a = np.array([1, 2, 3])
arr_b = np.array([4, 5, 6])
concatenated = np.concatenate([arr_a, arr_b])
print("Concatenated Array:", concatenated)
```

```
Concatenated Array: [1 2 3 4 5 6]
```

```
# Use np.unique to Find Unique Elements
arr = np.array([1, 2, 2, 3, 3, 3])
unique_elements = np.unique(arr)
print("Unique Elements:", unique_elements)
```

```
Unique Elements: [1 2 3]
```